Final Review Presentation

Cellular Automata and Algorithmic Preprocessing to Improve Clustering

Guide

Kamalika Bhattacharjee

Dr. Kamalika Bhattacharjee - Computer Science and Engineering

Team 8

G. Sneha

Vsubramanian

Ganta Sneha Rao - 106119037

Subramanian V V - 106119123

Viswonathan Manoranjan - 106119145

Introduction

- With Reversible Cellular Automata acting as natural clusters, we explore the possibility of using reversible CA rules and taking a non-conventional approach towards clustering numerical and categorical datasets, by grouping similar data points based on the rule and cycle formation using cellular automata.
- The goal is to propose a suitable encoding for the dataset, provide an efficient algorithm for clustering the data using the available CA rules and package our method for ease of access and use.

Literature Survey

- **Reversible Cellular Automata (RCA) clustering**

1. The reversibility of certain Cellular Automata served as our motivation to explore it as a clustering method for big datasets.
2. The properties of Reversible Cellular Automata are exploited for grouping the entities with minimum intra-cluster distance while ensuring that a limited number of cycles exist in the configuration space
3. Algorithms of past research perform at par with other state-of-the-art algorithms. [1] [2] [3] [4] [5].

- **Encoding Techniques**

1. Real-valued data cannot be processed by a CA. Thus, datasets need to be encoded to binary
2. Experimented with encoding algorithms previously used to encode datasets.
   a. Gödel number encoding [6] assigns a specific number to each symbol and operation within the expression, and combines them using a prime factorization algorithm.
   b. Entropy-based dimensionality reduction reduces the number of features in high-dimensional data by discarding the least informative ones based on their entropy [7] followed by hashing the numbers using Python's inbuilt hash function SipHash [8] helps reduce number of features and in turn length of binary string
   c. Other custom encoding techniques were experimented with, involving taking the mean and standard deviation of the data points to aggregate into features of less degree.

Limitations of Previous Work

a. Previous algorithms for RCA based clustering exhaustively search for the rules that give best results, leading to extremely long run times.
b. Previous RCA based clustering algorithms are restricted to a fixed window size, cluster size, and not packaged for ease of usage and access.
c. Godel number encoding [6] is infeasible as our algorithm's complexity is proportional to the length of the encoded string, and Godel numbers are huge in size
d. Entropy-based dimensionality reduction [7] followed by hashing the numbers using Python's inbuilt hash function SipHash [8] results in similar data points transforming into vastly different strings.
e. Other custom encoding techniques were used, resulting in a long string, increasing our complexity.

Objectives/Contributions

Proposed a new encoding algorithm to avoid loss in data and improve the clustering silhouette score.

Proposed a new rule search algorithm that avoids the exhaustive search for rules for a particular dataset.

Packaged the whole process for ease of replication and development.

Provided flexibility and efficiency for multiple datasets with varying number of records and features by saving their states for ease of computation when used again.

Block Diagram

Data Collection

(Different datasets are explored and collected to ensure the algorithm works on all kinds of datasets.)

Data Pre-Processing

(All raw datasets are converted to numerical data.)

Data Encoding

(The pre-processed dataset is encoded into a binary string using fixed-width BiNCE encoding.)

Rule Filtering using Cycle Numbers

(All the cycles for each reversible rule is generated for each split and rules with the smallest number of cycles are selected.)

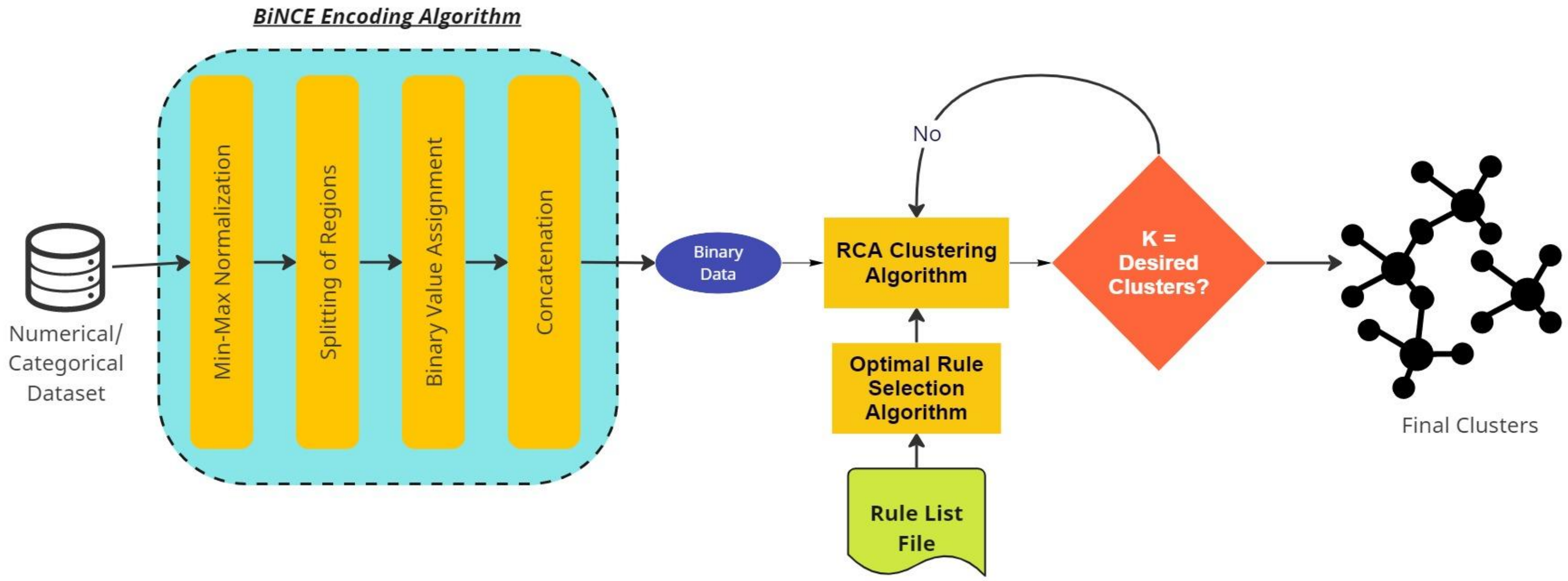
Reversible CA Clustering Algorithm

(The encoded binary string is split into chunks and the filtered CA rules are applied till the desired number of clusters are obtained.)

Analysis of the Results

(The best score out of 10 trials [by default] is displayed. Theories claimed earlier are proven by results.)

Data Flow Diagram



Algorithm of Modules

Encoding Stage

We use our novel **Binary Normalised Ceiling Encoding (BiNCE)** algorithm, a fixed-width encoding technique to encode numerical and categorical datasets into binarized datasets.

Step 1: Normalize the dataset using Min-Max normalization to scale the score between 0 and 1

Step 2: Given the bit size of encoding for each feature, split the 0-1 region into the corresponding parts. For eg. if 2 bit encoding is used, the 0-1 region is then separated into $2^2 = 4$ regions i.e. 0-0.25, 0.25-0.5, 0.5-0.75 and 0.75-1. Similarly, for n-bit encoding, the 0-1 region is split into n regions.

Step 3: Each of these regions is assigned a n-bit binary code in ascending order of value. For eg.
0-0.25 : ***00***
0.25-0.5 : ***01***
0.5-0.75 : ***10***
0.75-1 : ***11***

Step 4: The features are then concatenated horizontally to get a bitstring representing one single data. Length of the bitstring will be ***n*l***, *where* ***n*** *is the* ***bit size of each feature*** *in step 2, and* ***l*** *is the* ***number of features*** *of each data.*

Algorithm of Modules

Rule Selection Algorithm

Rule	Number of cycles
3537028050	4
3537035730	8
4031508660	8
3027809415	10
3537972705	10

Previous algorithms: Exhaustively apply all rules in search space to get the best score.

Our Rule Selection Algorithm,

Let $\boldsymbol{n}$ be the size of the vertical split, $\boldsymbol{R}$ be the rule set for a window size $\boldsymbol{w}$, and $\boldsymbol{N_{cyc\_a}}$, $\boldsymbol{N_{cyc\_b}}$ be the number of cycles of $\boldsymbol{r_a}, \boldsymbol{r_b} \in \boldsymbol{R}$.

> **Property** : In a given rule set $\boldsymbol{R}$, there exists $\boldsymbol{r_a} \in \boldsymbol{R}$ with $\boldsymbol{N_{cyc\_a}}$ which performs better at clustering for a Reversible Cellular Automata, compared to $\boldsymbol{r_b} \in \boldsymbol{R}$ with $\boldsymbol{N_{cyc\_b}}$, *where* $\boldsymbol{N_{cyc\_a}} < \boldsymbol{N_{cyc\_b}}$

This reduces our computation time exponentially as the total number of trials reduces from $\boldsymbol{{}^{162}C_2 = 13041}$, to $\boldsymbol{{}^{20}C_2 = 190}$, and in some cases even lower if initial clusters are less than desired clusters.

Number of rules	Combinations	Trials	Time Of Execution
162	${}^{162}C_2$	**13041**	3.6 Hours
20	${}^{20}C_2$	**190**	3.167 mins

* Chosen experimentally to balance the trade-off between speed and accuracy

Algorithm of Modules

Clustering Stage

Stage 1: Vertical Splitting:

Here, each encoded binary string of size n is split into equal divisions while padding the last split.

Stage 2: First Level Clustering:

On applying these rules to each of the vertical split, we distribute the partitioned configurations into some preliminary cycles. The (partitioned) configurations under the same cycle form a unique cluster.

Stage 3: Getting Desired Number Of Clusters:

We obtain desired number of clusters by merging the clusters till desired number is reached. We iterate through the clusters to find the smallest cluster and added to other clusters. The best fit for each element is determined by the clusters which give the highest performance index scores [silhouette score]. This procedure is followed till we reach desired number.

Implementation/ Simulation Environment

Environment

- ❖ Python == 3.11.0
- ❖ Scikit-learn == 1.2.0
- ❖ Pandas == 1.5.2
- ❖ Numpy == 1.23.5

- Processor : Intel i7 11700K 16M Cache, up to 5.00 GH
- Graphics Card : NVIDIA GeForce RTX A5000 24GB GDDR6
- Disk : 1TB SSD

Datasets

1. School District Breakdown Dataset
2. Iris Dataset
3. Customer Credit Card Information Dataset
4. Wine Dataset
5. Customer Segmentation Dataset
6. Heart Failure

01

Python

For ease of understanding, functional programming, GPU compatibility and replicability of previous papers

02

Numpy and Pandas

For data handling and faster/parallel computations.

03

Sklearn

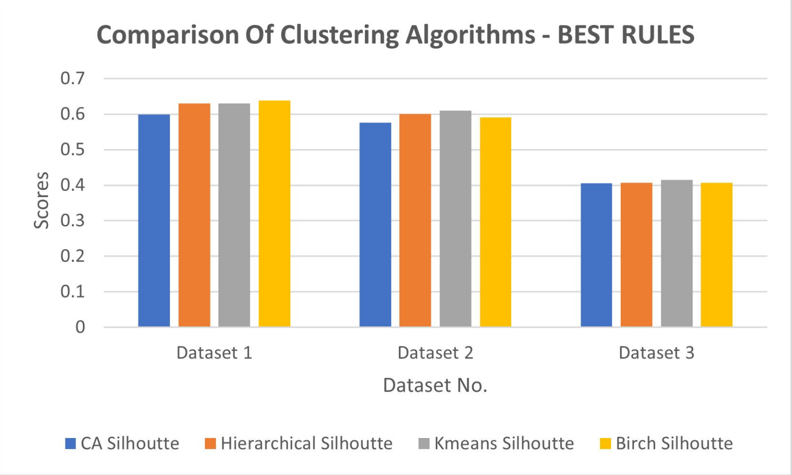
Employing K-Means, Birch and Agglomerative clustering for baseline scores and calculating silhouette scores

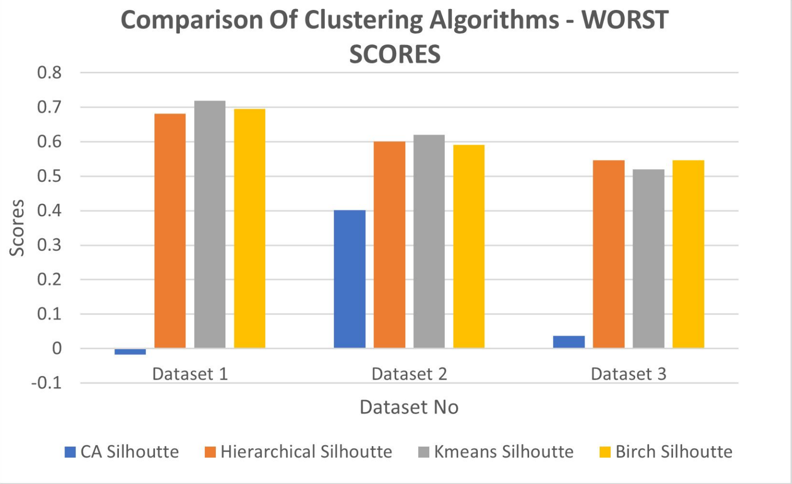
Results

Comparison of our CA clustering algorithm using filtered best and worst rules.

Best and Worst 5 results for Dataset 1 with split size 13

Best Rule Set			Worst Rule Set		
Silhouette Score	**Rules**	**Number Of Cycles**	**Silhouette Score**	**Worst Rules**	**Number Of Cycles**
0.59742392	3035673780	44	-0.0335495	254611245	3684
	3538955760	**16**		1259293455	3684
0.5366897	3035673780	44	-0.0534726	256577295	6096
	4027576560	64		1259293455	3684
0.5366897	3035673780	44	-0.0640834	755961615	5156
	3785744805	38		1924428408	5664
0.5217529	3035673780	44	-0.0534726	256577295	6096
	4042272240	44		1259293455	3684
0.5017529	3035673780	44	-0.0335495	254611245	3684
	517136850	**14**		1259293455	3684

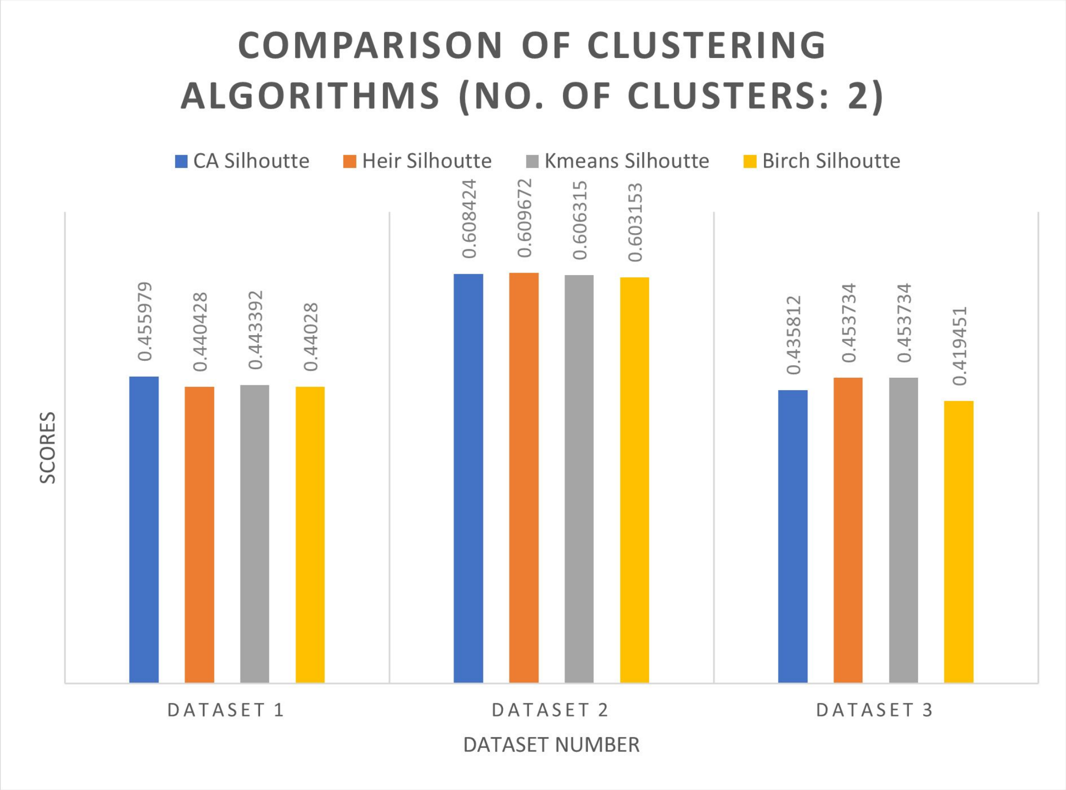




Results

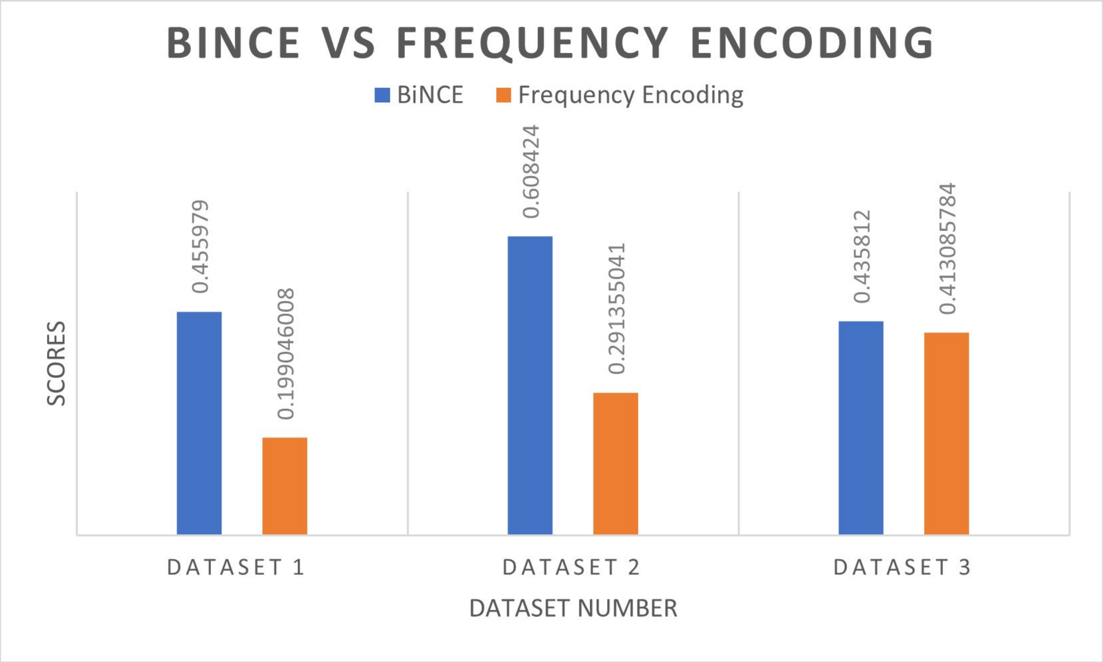
Comparison of our CA clustering algorithm with K-Means, Hierarchical and Birch clustering algorithms to give a better measure of its performance.

Credit Card Customer (Dataset 3)			
RCA	Hier	Kmeans	Birch
0.435812	0.453734	0.453734	0.419451
Customer Segmentation			
RCA	Hier	Kmeans	Birch
0.58454621	0.56822142	0.5834469	0.59377986
Heart Failure			
RCA	Hier	Kmeans	Birch
0.58162802	0.67892885	0.58288851	0.67892885
Wine Dataset			
RCA	Hier	Kmeans	Birch
0.46761462	0.6587293	0.65685365	0.6587293
IRIS Dataset (Dataset 2)			
RCA	Hier	Kmeans	Birch
0.60842444	0.60967234	0.6063151	0.60315354
SDB Dataset (Dataset 1)			
RCA	Hier	Kmeans	Birch
0.45597978	0.440428	0.443392	0.44028



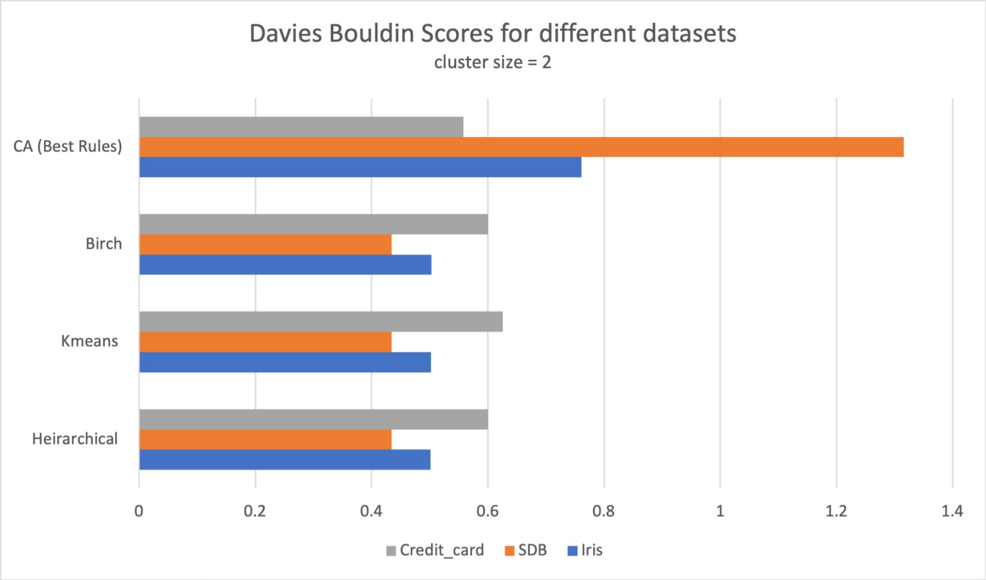
Results

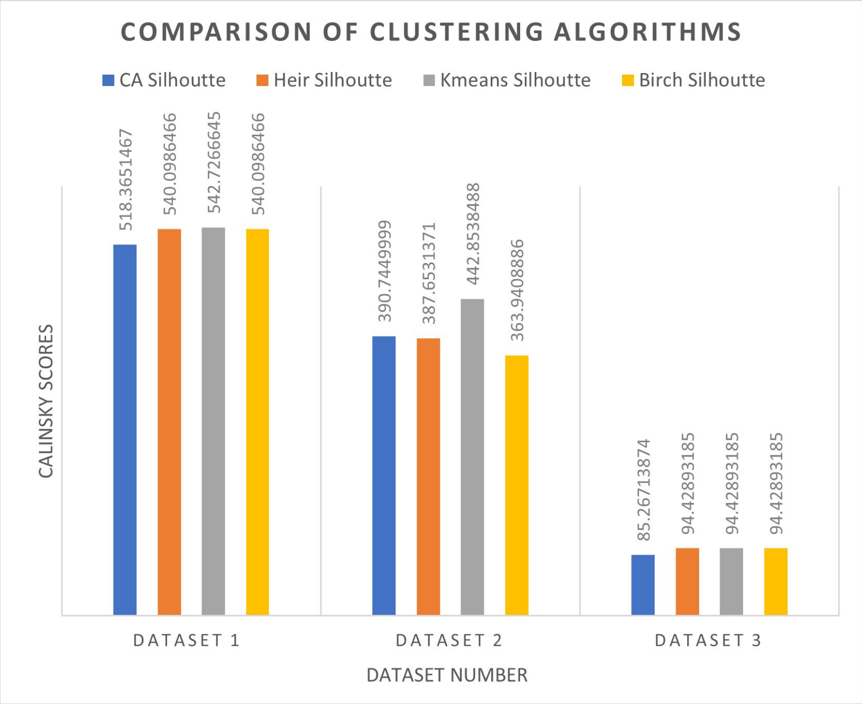
Compared to the previously used **Frequency Based Encoding**, the novel **BiNCE** encoding algorithm results in less loss of data during encoding numerical i.e. floating point and decimal datasets, hence leading to competitive scores with that of state-of-the-art clustering methods.



Results

Comparison of our CA clustering algorithm with K-Means, Hierarchical and Birch clustering algorithms using other clustering metrics





References

[1] *Sukanya Mukherjee, Kamalika Bhattacharjee, and Sukanta Das. (2020).* **Cycle Based Clustering Using Reversible Cellular Automata.** *In Cellular Automata and Discrete Complex Systems: 26th IFIP WG 1.5 International Workshop, AUTOMATA 2020, Stockholm, Sweden, August 10–12, 2020, Proceedings. Springer-Verlag, Berlin, Heidelberg, 29–42.*

[2] *S. Mukherjee, K. Bhattacharjee and S. Das*, "**Clustering Using Cyclic Spaces of Reversible Cellular Automata**," *Complex Systems, 30(2), 2021 pp. 205–237.*

[3] *Mukherjee, Sukanya & Bhattacharjee, Kamalika & Das, Sukanta. (2021)*. "**Reversible Cellular Automata: A Natural Clustering Technique. Journal of Cellular Automata**". *16. 1-38.*

[4] *Bhattacharjee, Kamalika & Naskar, Nazma & Roy, Souvik & Das, Sukanta. (2020):* "**A Survey of Cellular Automata: Types, Dynamics, Non-uniformity and Applications**"*: Vol. 19, Natural Computing.*

[5] *Abhishek S, Mohammed Dharwish, Amit Das, and Kamalika Bhattacharjee*, "**A Cellular Automata Based Clustering Technique for High-Dimensional Data**", *Proceedings of the Second Asian Symposium on Cellular Automata Technology 2023, IIEST Shibpur*

[6] *Negadi, Tidjani:* **"The genetic code via Godel encoding"***: arXiv preprint arXiv:0805.0695,2008.*

[7] *Hong He, Yonghong Tan*, "**Automatic pattern recognition of ECG signals using entropy-based adaptive dimensionality reduction and clustering**", *Applied Soft Computing, Volume 55, 2017*

[8] *Jean-Philippe Aumasson and Daniel Bernstein. (2012)* **"SipHash: a fast short- input PRF"**. *In: vol. 7668*

Appendix

Packaging of algorithms as Python Modules/ Classes

```python
import numpy as np
import pandas as pd

class BiNCE:
    def __init__(self,dataset,length_of_each) -> None:
        self.dataset_name =  'encoded_db'
        self.dataset_path = "../../Dataset"
        self.df = dataset
        self.save_path = '.temp/Dataset/fixed_width_encoding_'+self.dataset_name+'.csv'
        self.length_of_each = length_of_each
        self.even_regions = pow(self.length_of_each, 2)

    # normalize the dataset to [0, 1]
    def minmaxscale(self,series):
        min_val = series.min()
        max_val = series.max()
        return (np.divide((np.subtract(series, min_val,  dtype=np.float32)), (np.subtract(max_val,min_val, dtype=np.float32))))

    def evenly_spaced_array(self,num_splits):
        return np.linspace(0, 1, num_splits+1)[1:]


    def get_binary(self, arr2, number):
        idx = np.where(arr2 == number)
        return format(idx[0][0], '0{}b'.format(self.length_of_each))


    # Function to find the next greater element in an array for a given element
    def next_greater_element(self, x, arr):
        mask = arr >= x
        if any(mask):
            return arr[mask][0]
        else:
            return np.nan

    def encode(self):
        for cols in self.df.columns:
            self.df[cols] = self.minmaxscale(self.df[cols])
            self.df[cols] = self.df[cols].fillna(0)
        self.df = self.df.to_numpy()
        arr2 = self.evenly_spaced_array(self.even_regions)
        # Applying the function to each element in arr1
        new_df = []
        next_greater = []
        for arr1 in self.df:
            next_greater_arr = np.array([self.next_greater_element(x, arr2) for x in arr1])
```

```python
import pandas as pd
import numpy as np
from pandas import DataFrame as df
import math, sys, os
import random as rand
from itertools import combinations, permutations
import pickle as pkl
import threading
from sklearn.metrics import silhouette_score,davies_bouldin_score,calinski_harabasz_score
from sklearn.cluster import KMeans, Birch
from sklearn import metrics
import copy
from numpy import random
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import AgglomerativeClustering
import matplotlib.cm as cm
from multiprocessing import Process
from tabulate import tabulate
import warnings
from bin.BiNCE_encoding import BiNCE

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
class RCAC:
  def __init__(self,Dataset_name,split_size,num_clusters,data_drop_columns, compare_with_others = False, trials = 10,num_threads = 4) -> None:
    ## Make changes only in this section. Preferably don't change path
    ## The code will automatically make the directories if they don't exist.

    self.save_location = '.temp/'
    self.trials = trials
    self.num_threads = num_threads
    self.labels_ = []
    self.best_so_far = 0
    self.Dataset_name = Dataset_name
    self.rule_kind = 'best'
    self.split_size = split_size
    self.num_clusters = num_clusters
    self.split_index = 0
    self.compare = compare_with_others
    self.data_drop_columns = data_drop_columns
    warnings.filterwarnings("ignore")


  # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

  def split_string(self, encodings, div):
    r_ind=[]
    enc_length = len(encodings[0])
    # div = math.floor((enc_length)/split)+1
    for i in range(div, enc_length, div):
      r_ind.append(i)
    iclust=[]
    for i in range(len(encodings)):
      s=0
      for j in r_ind:
        iclust.append(encodings[i][s:j])
        s=j
      iclust.append(encodings[i][s:])
      encodings[i]=iclust
      iclust=[]
    return encodings, len(r_ind)+1
```

Appendix

Rule application and Driver functions

```python
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

#Will apply the CA rule and generate all possible cycles and returns nested list
def apply_rule(self, split,winsize,brule):
  final_array = []
  split_list=list(set(split))
  split_list.sort(reverse=True)
  split_len=len(split_list[0])
  current_array=[]
  while(split_list):
    curr_element=split_list[0]
    flag=0
    while(not flag):
      if current_array == []:
        current_array.append(curr_element)
        split_list.remove(curr_element)

      else:
        curr_element=self.nullbound(split_len,winsize,curr_element)
        t2=""
        for j in range(split_len):
          check=int(str(curr_element[j:j+winsize]),2)
          t2+=brule[check]

        curr_element=t2
        if curr_element not in current_array:
          if curr_element in split_list:
            split_list.remove(curr_element)
            current_array.append(curr_element)
        else:
          flag=1

    final_array.append(current_array)
    current_array=[]

  return final_array

# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
```

```python
# # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

# Main function
def fit(self):
  rule_list_name = self.rule_kind+'_cycles_'+str(self.split_size)
  os.makedirs('./'+self.save_location+self.Dataset_name+'/Custers-'+str(self.num_clusters)+'/Final Clusters', exist_ok=True)
  os.makedirs('./config/'+self.Dataset_name+'/Custers-'+str(self.num_clusters), exist_ok=True)

  self.data=pd.read_csv("./data/"+self.Dataset_name+".csv")
  self.data=self.data.dropna()
  self.data=self.data.drop(self.data_drop_columns,axis=1)
  window_size = 5
  encoder = BiNCE(self.data, length_of_each=2)
  self.enc = encoder.encode()
  self.enc = list(self.enc[self.enc.columns[0]])
  print(self.enc)

  split_enc, num_of_splits = self.split_string(self.enc,self.split_size)
  rule_list = self.get_rule_list('./rules/'+rule_list_name+'.txt')
  rules_comb = list(combinations(rule_list, 2))

  thread_pool = []
  rules_comb = np.array(rules_comb)
  rules_comb = rules_comb[:len(rules_comb)]
  rules_sep = np.array_split(rules_comb,(self.num_threads))
  for iclust in range(self.num_threads):
    print('Thread : ',iclust)
    init_clusters_ = threading.Thread(target=self.cellular_automata_clustering, args=(num_of_splits, rules_sep[iclust], split_enc, iclust, self.trials, window_size))
    init_clusters_.daemon = True
    init_clusters_.start()
    thread_pool.append(init_clusters_)

  for iclust in thread_pool:
    iclust.join()
```

Appendix

GitHub Repository